

### ARGUMENTS/REMARKS

Applicants would like to thank the Examiner for the careful consideration given the present application. The application has been carefully reviewed in light of the Office action, and amended in response thereto.

Claims 1-17 remain in this application.

Claims 1-6 were rejected under 35 U.S.C. §103(a) as being unpatentable over Portuesi (U.S. 6,499,057) in view of the GIS article (Internet GIS and Its Applications in Transportation). For the following reasons, the rejection is respectfully traversed.

Claim 1 recites a “moving picture coding means for producing *compressed moving picture data* having a high image quality from the *still picture data* obtained from said picture data relaying means”. Claim 4 recites similar limitations at lines 10-12. Neither reference suggests any means of producing “compressed moving picture data” from “still picture data”. Thus, claim 1 is patentable over the references. Claims 2-3, which depend on claim 1, and claims 5-6, which depend on claim 4, are thus patentable over the references for at least the same reasons as their parent claims.

In addition, the QUICK TIME format discussed by taught by Portuesi is a standard that does not teach the “moving picture coding means” as recited in the claims. The invention utilizes moving picture data converted from a still picture by using moving picture coding means (see, e.g., claim 1, lines 13-21). In contrast, the QUICK TIME format is taught as utilizing still picture data coded from a still picture by still picture coding means (i.e., JPEG).

In particular, QUICK TIME is a bundle of tool sets consisting of components that each have a media processing function. Regarding a still picture, QUICK TIME retains a still picture data coded by still picture coding means in a QuickTime file, or refers the external data as reference information. QuickTime expands or plays such data by means of Image Decompression (e.g., JPEG decoding) which is one of the decompression components. The attached materials provided a discussion of the QuickTime tools.

Accordingly, even if the zoom feature of Internet GIS and QuickTime (as taught by Portuesi) were combined, that would not suggest the “moving picture coding means” of the

invention.

Furthermore, the Examiner has not provided the proper motivation for combining the references. The burden is on the Examiner to make a prima facie case of obviousness (MPEP §2142). To support a prima facie case of obviousness, the Examiner must show that there is some *suggestion* or *motivation* to modify the reference (MPEP §2143.01). The mere fact that references *can* be combined or modified, alone, is not sufficient to establish prima facie obviousness (*Id.*). The prior art must also suggest the *desirability* of the combination (*Id.*). The fact that the claimed invention is within the *capabilities* of one of ordinary skill in the art is not sufficient, by itself, to establish prima facie obviousness (*Id.*). Merely stating that the combination is "obvious" is not motivation.

The Examiner argues that the motivation is provided because it would enable Portuesi to have a zoom feature as taught by GIS, thereby "enabling the image provider to be notified and thus enabling the user to view a selected image in more detail so that the user may more finely review the image". However, this argument is nothing more than a conclusory benefit, and is not sufficient to support a prima facie case of obviousness. The Examiner must show a convincing reason that the reader would add that benefit to the base reference. This has not been done.

Accordingly, because no proper motivation has been provided, the rejection for obviousness is improper, and the rejection should be withdrawn.

Claims 7, 9-14, and 16-17 were rejected under 35 U.S.C. §103(a) as being unpatentable over Portuesi in view of GIS and further in view of Tracton (U.S. 6,470,378). Claims 8 and 15 are rejected as above in further view of Guedalia (U.S. 6,536,043). For the following reasons, the rejections are respectfully traversed.

Claim 7 recites similar limitations (at lines 7-8) as discussed for claim 1. Claim 13 recites similar limitations at lines 8-9, and claim 17 at lines 15-17. Neither Tracton nor Guedalia teach any means for producing compressed moving picture data from content data representing a high-quality still image. Accordingly, claims 7, 13, and 17 are patentable over the references. Claims 8-12, which depend, directly or indirectly, on claim 7, and claim 14, which depends on claim 13, are thus patentable over the references for the same reasons as the parent claims.

Furthermore, claim 13 recites that "image processing is conducted at the information providing system to thereby reduce a processing load and/or a memory requirement on the mobile terminal". None of the references suggest such a scheme.

Even further, claim 17 recites that the mobile terminal includes telephone capability, which is not suggested by any of the references.

Finally, the Examiner has again failed to provide legally sufficient motivation for combining the references, and thus the rejection is improper.

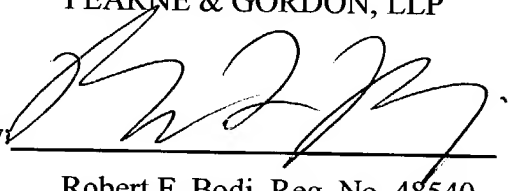
In consideration of the foregoing analysis, it is respectfully submitted that the present application is in a condition for allowance and notice to that effect is hereby requested. If it is determined that the application is not in a condition for allowance, the examiner is invited to initiate a telephone interview with the undersigned attorney to expedite prosecution of the present application.

If there are any additional fees resulting from this communication, please charge same to our Deposit Account No. 16-0820, our Order No. 32626.

Respectfully submitted,

PEARNE & GORDON, LLP

By

  
Robert F. Bodi, Reg. No. 48540

1801 East 9<sup>th</sup> Street, Ste. 1200  
Cleveland, Ohio 44114-3108  
(216) 579-1700

November 18, 2004



RECEIVED

NOV 29 2004

Technology Center 2600

Specific tasks that QuickTime is useful for include:

- Playing movies and other media, such as Flash or MP3 audio
- Nondestructive editing of movies and other media
- Importing and exporting images between formats, such as JPEG and PNG
- Compositing, layering, and arranging multiple media elements from different sources
- Synchronizing multiple time-dependent media to a single timeline
- Capturing and storing sequences from real-time sources, such as audio and video inputs
- Creating movies programmatically from synthesized data
- Creating sprites that use intelligent, scripted animation
- Creating presentations that interact with viewers, remote databases, and application servers
- Creating movies that include customized window shapes, "skins," and controls
- Streaming movies in real time over a network or the Internet
- Broadcasting real-time streams from live sources such as cameras and microphones
- Distributing downloadable media on disc or over a network or the Internet

RECEIVED  
DEC 13 2004  
TC 1700

## Architecture

---

The QuickTime programming architecture is a combination of flexible tool sets and plug-in components.

### Tool Sets

---

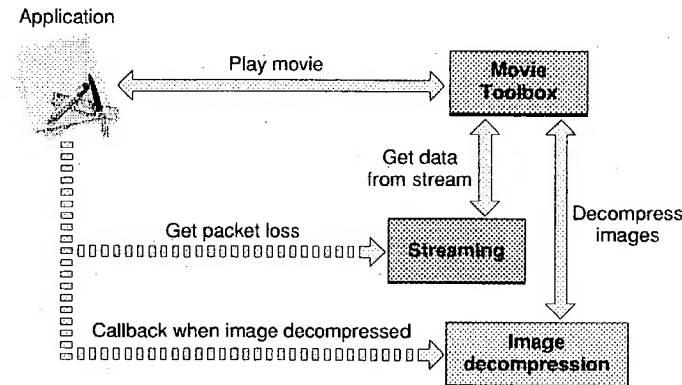
To support the complete spectrum of multimedia tasks, the QuickTime API contains a collection of **tool sets**, such as the **Movie Toolbox**, the **Image Compression Manager**, the **sequence grabber**, and the **QuickTime streaming API**.

- The Movie Toolbox is used to initialize QuickTime; open, play, edit, and save movies; and manipulate time-based media.
- The Image Compression Manager is a device-independent and driver-independent means of compressing and decompressing image data.
- The sequence grabber is a framework for components that capture and record samples from real-time sources, such as video cards or audio inputs.
- The streaming API allows you to send and receive real-time streams using standard protocols such as RTP and RTSP.

There are several other tool sets, including **QuickTime VR**, the **sprite toolbox**, and the **wired movies API**, but you don't need to use them all or even know them all. These tool sets work together, allowing you to focus on the task at hand, without needing to learn the entire QuickTime API. The different tool sets often share data types and programming paradigms, making it relatively easy to extend your knowledge of QuickTime as you go.

Many tool sets are useful when you need direct access to things that QuickTime usually deals with automatically. For example, when you use the Movie Toolbox to play a movie, it may open a stream of real-time data and decompress a series of images, without requiring you to interact with the streaming API or the Image Compression Manager. But if you need to check for streaming packet loss or be notified each time an image is decompressed, you can use the appropriate tool set from your application.

**Figure 1-1** Using tool sets



## Components

The QuickTime architecture makes extensive use of **components**, making it modular, flexible, and extensible. A QuickTime component is a shared code resource with a defined API. It is possible to add a new component to QuickTime and have existing applications automatically find it and use it when needed, largely because it responds to the same API as existing components of that general type.

For example, QuickTime works with a number of media types: sound, video, text, sprites, Flash, 3D models, photographic virtual reality, and others. Each media type is supported by a media handler component. The number and types of supported media are continually growing. You can add a new media type to QuickTime yourself by creating a new media handler component.

There are also component types for controlling and playing movies, importing and exporting media, compressing and decompressing images or sound, accessing data (from file systems, network servers, or blocks of memory), capturing sequences of digitized sample data, and so on. Here is a partial list:

- Movie controller components are used to play movies and can provide a standard user interface.
- Media handler components handle a particular type of media data, such as video, sound, Flash, or text.
- Data handler components access data from a particular kind of data source, such as local files, URLs, pointers, or handles.
- Image compressor components compress or decompress image data.
- Image compression dialog components let the user specify the parameters for compression operations.

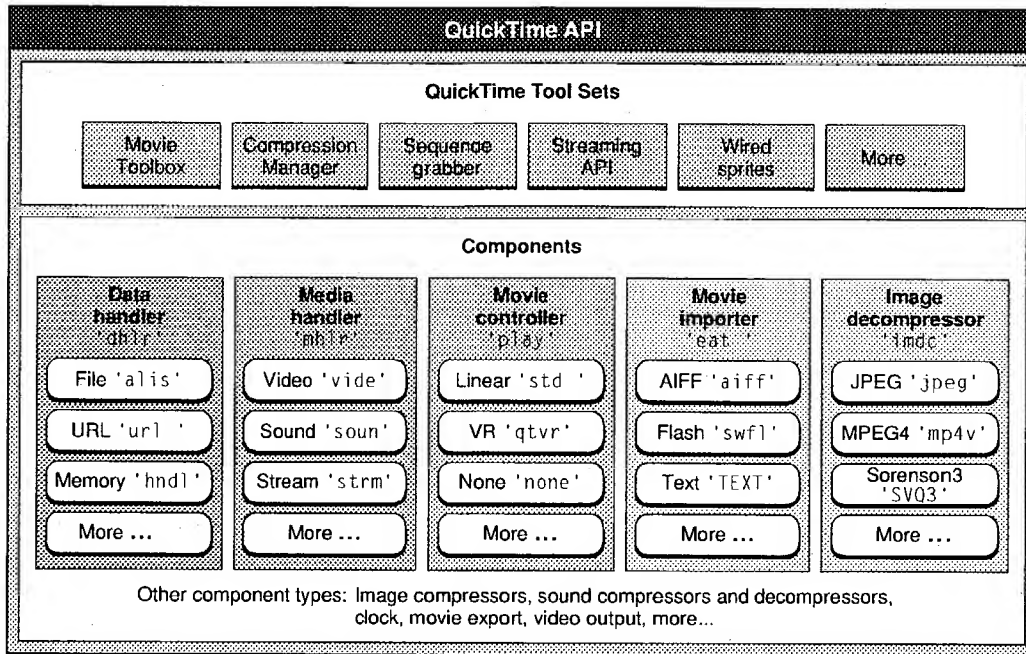
- Video digitizer components are used to control video digitization by external devices such as video capture cards.
- Movie data-exchange components (also known as movie import and movie export components) move different types of data into and out of QuickTime movies. QuickTime can play any type of media file for which it has an importer or create any type for which it has an exporter.
- Video output components convert the visual output of QuickTime movies into video streams for devices other than displays.
- Graphics import and export components provide a single API that lets you work with a wide variety of image file formats.
- Music components process and synthesize MIDI-like music tracks in QuickTime movies.
- Effects and transitions components implement visual filters, effects, and transitions. (Effects components are implemented as a special type of image compressor component.)

As with tool sets, you don't need to work with, or even know about, every type of component. Most components are used automatically as needed, but most also support an API that you can work with directly when you want to.

Each component has a **type**, a **subtype**, and a **manufacturer** code, each represented by a four-character code. A four-character code is a 32-bit value that is usually best interpreted as four ASCII characters. For example, an image decompressor component has a type of 'imdc'.

QuickTime often has multiple components of a given type. For example, QuickTime has many decompressor components. They all have the same type: 'imdc'. Each 'imdc' component has a subtype that specifies the kind of compression it understands, such as JPEG, and a manufacturer code that distinguishes among components of the same subtype. For example, the image decompressor for JPEG supplied by Apple has the type, subtype, and manufacturer codes of 'imdc', 'jpeg', 'aapl'.

Figure 1-2 Some commonly used tool sets and components



**Important:** There are no "three-character" codes. Some of the four-character codes, such as 'eat', include an ASCII blank space (0x020). The space character is an important part of the code, and cannot be omitted.

QuickTime ships with a number of components and has the ability to download others from Apple's servers when needed (provided there is an Internet connection and user consent). Third-party components can be installed locally, and in many cases can be recognized and used by existing applications without modification. (Most third-party components are not automatically downloadable from Apple's servers, however.)

QuickTime finds, selects, loads, and unloads components as needed. This is often transparent to the applications programmer. For example, when you tell QuickTime to open a movie, QuickTime automatically finds and loads the correct media handlers and decompressors for the movie. An error is returned if these operations fail, but otherwise they are transparent to the application.

Nearly all QuickTime programmers need to deal directly with components from time to time, however. For example, to play a QuickTime movie, most applications work directly with a movie controller component.

Selecting and working with components is documented in *Component Manager for QuickTime*. This is a subset of the Component Manager documentation for the Mac OS (Carbon); it describes the parts of the Component Manager that QuickTime programmers are likely to use, and that are included in QuickTime for Windows. All QuickTime programmers should read *Component Manager for QuickTime*.

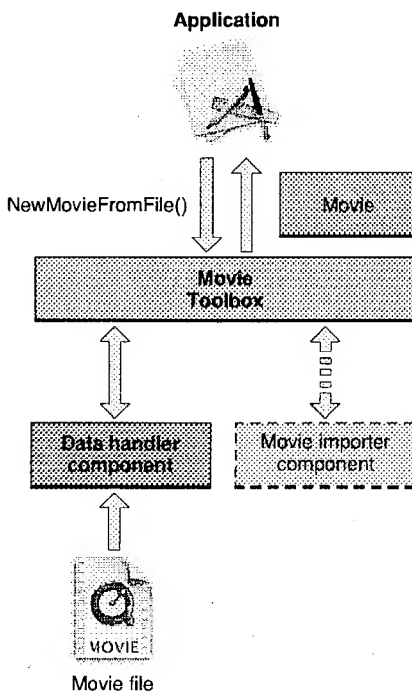
Specific QuickTime components are discussed in the QuickTime documentation for the topic relevant to that component. For example, movie controller components are described in *Opening and Playing Movies* and compressor components are described in *"Compression and Decompression"*.

Note that the primary QuickTime documentation describes the interface to components from the perspective of an application calling the component. There is an additional set of documents that describe writing new components (see *"Creating QuickTime Components"*). If you are writing a QuickTime component, you need to read the primary documentation for that component, the generic documentation for creating QuickTime components, and any additional documentation for creating that particular kind of component in order to understand how the component is used, what API you need to support, and how best to implement it.

## Examples

Let's look at two examples of the QuickTime architecture in action: getting a movie from a file and playing a movie using a movie controller.

**Figure 1-3** Getting a Movie

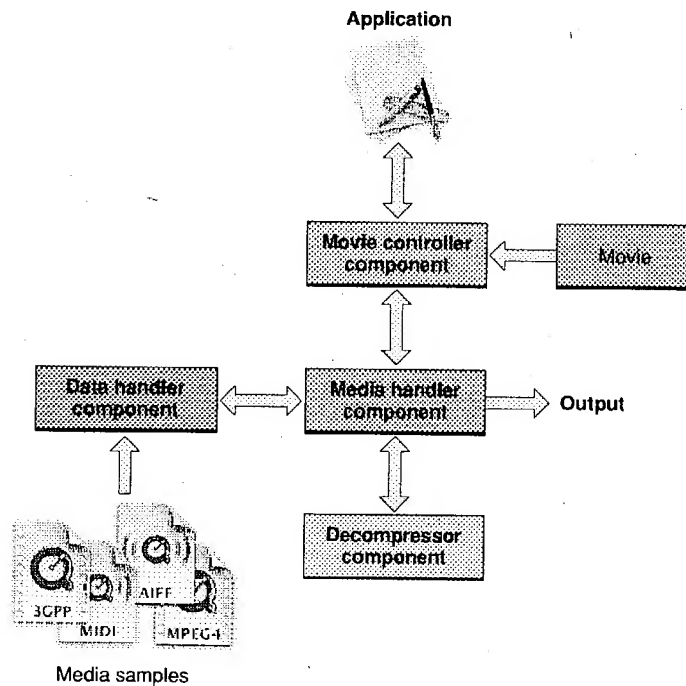


- Your application tells the Movie Toolbox to get a movie, in this case from a file.
- A data handler component is used to access the file; a different component would be used to get a movie from a local file, a URL, or a stream. QuickTime chooses the appropriate component based on the data source.



- If the file is not a QuickTime movie file, a movie importer component is used to create a movie from the file; a different component would be used to import from an MP3 file or a JPEG file. QuickTime chooses the appropriate importer based on the file type, file extension, or MIME type.
- The Movie Toolbox passes a movie to your application.

Figure 1-4 Playing a Movie



- Your application attaches a movie controller to the movie and the user presses the "Play" button on the control bar.
- A media handler component is used to work with each type of media used in the movie; different components are used for sound, video, Flash, and so on. QuickTime chooses the appropriate media handlers based on the media types.
- Media handlers makes calls to data handlers to access their media samples, which may come from a different data sources. For example, a movie on a local disk might point to media samples on a local disk, a remote file server, and an Internet stream. QuickTime chooses the appropriate data handler for each data source.
- A media handler typically makes calls to a decompressor component to decompress the media samples; different components are used for different media types, such as sound and video, and for different compression schemes, such as MP3 or MP4 audio, JPEG or GIF images. QuickTime chooses a decompressor based on the media type and compression scheme.
- A media handler may then pass its output directly to a low-level service, such as the Sound Manager, for final disposal, or to another component, such as a video output component, for further processing.

## Output

---

QuickTime can provide various kinds of output during playback. The output is typically sound and video, with a visible controller, but it can be simply sound, with no display or visible controller, or even a silent and invisible DV stream to a FireWire port.

The actual output is handled by a lower-level technology, such as QuickDraw, OpenGL, the Sound Manager, Core Audio, or DirectX. QuickTime shields you from having to deal with these details in most cases. When you play a movie, QuickTime selects and configures the default devices for playback on your platform.

Typically, QuickTime sends audio output to the Sound Manager (included in QuickTime for Windows), or to Core Audio in Mac OS X. Similarly, the visual output from QuickTime is typically sent to a graphics port or graphics world (GWorld), which relies on QuickDraw (the necessary parts of which are included in QuickTime for Windows) or sent to a graphics context managed by Quartz (Mac OS X only). QuickTime can be specifically directed to send its visual output to any device that has an installed video output component, however, and you can create a context to specify a particular output technology, such as Core Audio or OpenGL.

Most QuickTime programmers need to learn a little about graphics worlds and QuickDraw, and may need to refer to "Color QuickDraw", "Graphic Devices" or *QuickDraw Reference* to learn how to properly set up a graphics world, associate it with a graphics port, and work with the common QuickDraw data types, such as a rect.

Those wishing to control QuickTime output directly need to learn about their chosen output technology, such as Sound Manager, Core Audio, or OpenGL.

## The QuickTime API

---

The QuickTime API allows you to add a host of multimedia features to your application without needing to master the often arcane details of particular media formats and specifications.

For example, you can use QuickTime to open and display a series of JPEG images, concatenate them into a time-based slideshow with an MP3 sound track, then export the images as TIFF or PNG graphics, or export the slides and music together as a DV stream, without needing to learn the intricacies of the compression, stream, or file formats for JPEG, TIFF, PNG, MP3, or DV.

## Multilevel API

---

The QuickTime API includes over 2500 functions, divided into tool sets for particular tasks, with special functions and data types for virtually any task.

This can be a little daunting for programmers new to the API. It is easy to get lost in details and lose sight of the big picture. The most common error among new QuickTime programmers is to attack a problem using a complex, low-level tool set when there is a much simpler high-level command to perform the entire task.

You can interact with the QuickTime API at many different levels: